Off-chain Protocols: (Virtual) State Channels

Sebastian Faust

Credits: Stefan Dziembowski, Lisa Eckey, Julia Hesse, Kristina Hostakova, Sebastian Stammler
Hot topic in cryptocurrencies and academia

“Channels” is our main focus in this talk.

Initiated by [Decker & Wattenhofer] [Poon & Dryja] (plus many informal online publications).

“Plasma” initiated by [Poon & Buterin] (plus countless informal online publications).
Many interesting state channel projects

We follow the terminology of **Perun**
(L4 Counterfactual, Connext und Magmo are very similar)
Main goal: off-chain contracts

**On-chain contract:** deployment and execution on-chain

**Off-chain contract:** deployment and execution off-chain
State Channel Variants

**Goal:** Off-chain execution of contracts

- **Ledger state channel:** channel built „over ledger“
- **Virtual state channel:** channel built „over ledger channels“
- **Multiparty state channels:** channels for multiparty contracts

This talk
Outline

1. Introduction
2. Ledger Channels
   a. Recap: Ledger Payment Channels
   b. Ledger State Channels
3. Virtual State Channels
4. Security analysis
5. Summary
Main ingredient of off-chain protocols

**Smart contract** $\approx$ „programmable money“

**Examples:** Ethereum

1. Parties deploy contract and deposit coins to the contract
2. Execute the contract
3. Coins can be assigned back to the users
Ledger Payment Channels

**Goal:** Execute payments off-chain directly over ledger

**Examples:** Raiden Network in Ethereum, Lightning Network in Bitcoin
**Updating** the ledger payment channel

**Goal:** Update to new balance $0.99$ for Alice and $1.01$ for Bob

**Idea:** exchange signatures on message containing new balance

$$m := (0.99, 1.01, 1)$$

New balance of the parties

An index called version number

$$\sigma_{\text{Alice}} := \text{Sign}_{\text{Alice}}(0.99, 1.01, 1)$$

$$\sigma_{\text{Bob}} := \text{Sign}_{\text{Bob}}(0.99, 1.01, 1)$$
Further updates

For each update increase version number: $\text{version} = \text{version} + 1$

$\sigma_{\text{Alice}} := \text{Sign}_{\text{Alice}}(0.98, 1.02, 2)$

$\sigma_{\text{Bob}} := \text{Sign}_{\text{Bob}}(0.98, 1.02, 2)$
Closing the payment channel

Suppose Bob wants to close the channel

Alice gets notified that close was sent

Alice has 1 hour time to react

0.98 coins

Payment channel contract

Close(0.98, 1.02, 2, $\sigma_{Alice}$)

Verify $\sigma_{Alice}$ and pay out balance

1.02 coins

1 hour = maximal time needed to post on the ledger
Handling disputes

Suppose malicious Alice closes channel with earlier balance

Payment channel contract

\[
\text{Close}(0.99, 1.01, 1, \sigma_{\text{Bob}}) \\
\text{Close}(0.98, 1.02, 2, \sigma_{\text{Alice}})
\]

Verify \(\sigma_{\text{Alice}}\) and \(\sigma_{\text{Bob}}\)

Pay out balance according to highest version number

0.98 coins → 1.02 coins
Outline

1. Introduction
2. Ledger Channels
   a. Recap: Ledger Payment Channels
   b. Ledger State Channels
3. Virtual State Channels
4. Security analysis
5. Summary
State channels – motivation

One way to look at the payment channels:

Alice and Bob maintain a simulated ledger between themselves.

The contents of the “simulated ledger” is registered on the “real ledger” only if:

• the channel is closed, or
• the parties run into a dispute.

A natural question: can we also have contracts “inside of the simulated ledger”?
Ledger State Channels

**Goal:** Execute contracts off-chain directly over ledger
Add a contract

Recall: Contracts own coins and maintain state

Idea: Extend signed tuple by coins and state of contract

\[ m := (x_A, x_B, y, state, version) \]

Alice’s coins

Bob’s coins

Contract’s coins

Contract’s off-chain state

\[ y = y_A + y_B \]

\[ \sigma_{Alice} = \text{Sign}_{Alice}(m) \]

\[ \sigma_{Bob} = \text{Sign}_{Bob}(m) \]
Executing a contract

Suppose Alice wants to execute contract on function \textit{Move} with input \textit{a}

How can a contract execution be done in a \textit{state channel}?  

1. \textbf{Peaceful execution}: optimistic (\textit{off-chain})  
2. \textbf{Forceful execution}: in case of dispute (\textit{on-chain})
Peaceful execution

Idea: Executing the contract in the state channel

Suppose latest state was: \((x_A, x_B, y, state, version)\)

Both parties locally compute:

Parties exchange signatures on: \((x_A, x_B, y, state', version+1)\)
**Forceful execution**

**Idea:** If dispute during execution occurs, parties move on-chain

Suppose latest state was: $(x_A, x_B, y, state, version)$

Register latest state $(x_A, x_B, y, state, version)$ in channel contract

*From then on:* continue execution on-chain
Security and efficiency guarantees

Consensus on create & add: Agreement required from Alice & Bob

1. Create: Contract requires confirmation from both parties
2. Add: Protocol requires signature from both parties

Guaranteed execution: Once contract is added, its execution cannot be halted
⇒ achieved via forceful execution

Optimistic execution in O(1) rounds: When both parties honest contract execution is in real time
⇒ achieved via peaceful execution
Summary – ledger channels

Main features compared to on-chain transactions

- **Low costs**
- **Instantaneous updates/execution**

<table>
<thead>
<tr>
<th></th>
<th>Payment channels</th>
<th>State channels</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Purpose</strong></td>
<td>2-party payments</td>
<td>2-party contracts</td>
</tr>
<tr>
<td><strong>Create channel</strong></td>
<td>On-chain</td>
<td>On-chain</td>
</tr>
<tr>
<td><strong>Update/Execute</strong></td>
<td>Off-chain balance changes</td>
<td>Off-chain contract execution</td>
</tr>
<tr>
<td><strong>Close</strong></td>
<td>On-chain</td>
<td>On-chain</td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. Ledger Channels
   a. Recap: Ledger Payment Channels
   b. Ledger State Channels
3. Virtual State Channels
4. Security analysis
5. Summary
Channel networks

Problem: every pair of parties requires a separate channel

Can we do better? Yes!

Option 1: Routing each payment via Ingrid
  • Hashed Timelock Contract (HTLC): used for payments in e.g., Lightning

Option 2: Creating virtual state channels
  • Used for state channel networks
Virtual state channel – overview

Bob, let's play!

1. Create
2. Add
3. Execute
4. Close
Virtual state channel: create

Ingrid puts collateral $0.2$ to take role of Bob

Virtual Channel Contract (VCC)

Ingrid puts collateral $0.1$ to take role of Alice
Virtual state channel: create

Ingrid agreed

Ingrid puts collateral 0.2 to take role of Bob

Virtual Channel Contract (VCC)

Ingrid agreed

Ingrid puts collateral 0.1 to take role of Alice
Virtual state channel: create

Summary: Use sub-protocol „add“ of underlying state channel to add contract representing virtual state channel
Virtual state channel: contract lifecycle

1. Adding contract
2. Peaceful execution of contract
3. Closing contract

Exchanging signatures on new version tuples between parties
Virtual state channel: close

Before close: Alice gained 0.1 coins and Bob lost 0.1 coins

After close:
- Alice gained 0.1 coins
- Financially neutral
- Lost 0.1 coins
So far: All parties behave honestly

How to handle malicious parties?

**Forceful execute** protocol in virtual channels
(Two methods: indirect vs. direct disputes)
Virtual state channel: **forceful execute**

**Forceful execute via indirect dispute:** dispute resolution via intermediary

- **Start registration**
- **Bob is not responding**
- **Move registration on-chain**
Virtual state channel: **forceful execute**

**Forceful execute via indirect dispute:** dispute resolution via intermediary

- **Advantage:** Alice does not need to go on-chain
- **Disadvantage:** Longer time for dispute resolution
Virtual state channel: **forceful execute**

**Forceful execute via direct dispute:** dispute resolution directly on blockchain

- **Advantage:** Conflicts can be resolved faster
- **Disadvantage:** Honest Alice needs to go to ledger
Extension: longer channels

Idea: Underlying (ledger/virtual) state channels are used as building block
Two additional **security** and **efficiency** guarantees

**Balance neutrality:** Ingrid never looses money

Whatever malicious Alice and Bob do, Ingrid will always receive back her 2 coins

**Pessimistic execution** for virtual channels built over n ledger state channels:
- Direct dispute: $O(n)$ rounds
- Indirect dispute: $O(1)$ rounds
Summary – ledger vs. virtual

Technique to connect ledger channels
• Used mainly for state channel networks

Minimize interaction with intermediary

<table>
<thead>
<tr>
<th></th>
<th>Ledger channels</th>
<th>Virtual channels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>On-chain</td>
<td>Off-chain</td>
</tr>
<tr>
<td>Update/Execute</td>
<td>Off-chain</td>
<td>Off-chain</td>
</tr>
<tr>
<td>Close</td>
<td>On-chain</td>
<td>off-chain</td>
</tr>
</tbody>
</table>
Outline

1. Introduction
2. Ledger Channels
   a. Recap: Ledger Payment Channels
   b. Ledger State Channels
3. Virtual State Channels
4. Security analysis
5. Summary
Complex protocols
How to analyze complex protocols
One approach – Modern cryptography

Provable security: Sound methodology for complex protocols

Define security of protocol → Design protocol → Prove security

Two ways of defining security:

- **Game-based**: adversary violates certain security guarantees. Example: IND-CPA security for encryption

- **Simulation-based**: Adversary in real-world cannot do "better" than in ideal world. Example: crypto protocols in the universal composability model.
The UC model – Environment

Describes “everything else” happening in the world (e.g., other protocol executions etc.)

Environment orchestrates:

• The **parties** by providing **inputs** to and receives outputs from them

• The **adversary** that can corrupt parties and communicate with environment in each step of protocol execution via **side effects** (communication, consumed time and resources, etc.)

Analyse security of protocol by analysing its effects on the environment
The UC model – The Ideal world

How to analyse the effects that a protocol has on its environment?

• Define an **ideal functionality** $\mathcal{F}$ capturing “ideal effects” on the environment
• Construct **simulator** that simulates “real effects” on the environment
• Show that $P$ and $(\mathcal{F}, \mathcal{S})$ “look the same” for any environment $\mathcal{E}$
The UC model – The formal security definition

$P$ UC-realizes $\mathcal{F}$ if $\exists \mathcal{S}$ s.t. $\forall \mathcal{E}$ the interaction with $(\mathcal{F}, \mathcal{S})$ is indistinguishable from the interaction with $P$

Might be difficult to prove if $P$ very complex

Technique: Modularize protocol and proofs
The UC model – Simplifying via hybrids and composition

<table>
<thead>
<tr>
<th>Ideal world</th>
<th>Hybrid world</th>
<th>Real world</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Ideal world diagram" /></td>
<td><img src="image" alt="Hybrid world diagram" /></td>
<td><img src="image" alt="Real world diagram" /></td>
</tr>
</tbody>
</table>

Show hybrid protocol \((\rho, \mathcal{H})\)

UC-realizes ideal functionality \(\mathcal{F}\)

Show that \(\pi\) UC-realizes \(\mathcal{H}\)

Modularize protocol \(P\):

\(P\) composed of \((\rho, \pi)\) with \(\rho\) the main protocol and \(\pi\) sub-routines
The UC model – Overview of approach

1. Ideal functionality $\mathcal{F}$
   - Abstract specification

2. Protocol $\mathcal{P}$
   - Define behavior of each party

3. Simulator $\mathcal{S}$

4. Prove indistinguishability
   - $\forall$ environment $\mathcal{E}$, real and ideal world indistinguishable

<table>
<thead>
<tr>
<th>Ideal world</th>
<th>Real world</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{F}$</td>
<td>$\mathcal{P}$</td>
</tr>
<tr>
<td>$\mathcal{S}$</td>
<td>$\approx$</td>
</tr>
<tr>
<td>$\mathcal{E}$</td>
<td>$\mathcal{E}$</td>
</tr>
</tbody>
</table>
The UC model for ledger based protocols

How to model a public ledger in UC?

- Global ideal functionality
  - List of accounts
  - Everyone can read its state
    - Also the environment!

- State must be the same in real and ideal world

<table>
<thead>
<tr>
<th>Ideal world</th>
<th>Real world</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ledger</strong></td>
<td><strong>Ledger</strong></td>
</tr>
<tr>
<td>Alice 5</td>
<td>Alice 5</td>
</tr>
<tr>
<td>Bob 6</td>
<td>Bob 6</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

\[
\mathcal{F}_{ch}
\]

input \rightarrow output

input \rightarrow output

output \rightleftarrows output
1. State Channels – Define ideal functionality

## Ledger

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>5</td>
</tr>
<tr>
<td>Bob</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

## State Channel Functionality

- Create
- Add
- Execute
- Close
2. State Channels – Design protocol

**State Channel Smart Contract**
- Modelled as an ideal functionality
- Hybrid functionality of the protocol

**Ledger**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>5</td>
</tr>
<tr>
<td>Bob</td>
<td>6</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

State channel smart contract hybrid world
3. State Channels – Build simulator

Main tasks:

- Changes on the ledger
  - Same amount
  - At the same time

- Outputs
  - Same content
  - At the same time

<table>
<thead>
<tr>
<th>Ideal world</th>
<th>Hybrid world</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ledger</strong></td>
<td><strong>Ledger</strong></td>
</tr>
<tr>
<td>Alice 4</td>
<td>Alice 4</td>
</tr>
<tr>
<td>Bob 6</td>
<td>Bob 6</td>
</tr>
</tbody>
</table>

\[
\mathcal{F}_{ch}
\]

\[
S
\]

\[
\mathcal{T}_{transact}
\]

\[
\mathcal{T}_{output}
\]
3. State Channels – Simulator

Main tasks:
- Changes on the ledger
  - Same amount
  - At the same time
- Outputs
  - Same content
  - At the same time

<table>
<thead>
<tr>
<th>Ideal world</th>
<th>Hybrid world</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ledger</strong></td>
<td><strong>Ledger</strong></td>
</tr>
<tr>
<td>Alice 4</td>
<td>Alice 4</td>
</tr>
<tr>
<td>Bob 6</td>
<td>Bob 6</td>
</tr>
</tbody>
</table>

Even if parties are corrupt
4. State Channels – Proofs using modular approach

Recall: protocol design

Proof steps
1. Channel of length $i$ build on top of channels of length $\leq i - 1$
2. UC Composition
3. Channel of length 1 build using state channel smart contract
4. UC Composition
Outline

1. Introduction
2. Ledger Channels
   a. Recap: Ledger Payment Channels
   b. Ledger State Channels
3. Virtual State Channels
4. Security analysis
5. Summary
Summary

1. Use cases require **different** off-chain solution

2. **Much more research** that was not covered in this talk: Multiparty state channels, Services, Privacy, Routing, Optimizations,…

3. **Future research directions:**
   a) More formal modelling and proofs needed
   b) Hybrids of Channels & Plasma
   c) Explore limitations via lower bounds
   d) Optimization and implementation
   e) ...
Thank you for your attention!

Icons made by Freepik, Smashicons, and Nikita Golubev from www.flaticon.com